Homework 3

Andrew Patella

February 7, 2025

1 Consider the equation $2x - 1 = \sin x$

1.1 Find a closed interval $[a, b]$ on which the equation has a root r, and use the Intermediate Value Theorem to prove that r exists.

Rearrange:

$$2x - 1 - \sin x = 0 = f(x)$$

This new form of the equation allows us to use the IVT to solve for roots. By finding two values of x that swap signs, we can determine that a root must exist.
Try $x = 0$:

$$f(0) = 0 - 1 - 0 < 0$$

Try $x = \pi$:

$$f(\pi) = 2\pi - 1 - 0 > 0$$

So on the interval $x \in [0, \pi]$, there exists some root. This is because the function $f(x)$ is continuous on this interval, and it changes signs from x_1 to x_2. By the IVT, there must be at least one root in $x \in [0, \pi]$.

1.2 Prove that r from (a) is the only root of the equation on all of the real numbers.

The derivative of f will be used to determine if multiple roots exist.

$$f(x) = 2x - 1 - \sin x$$
$$f'(x) = 2 - \cos x$$

Since $|\cos x| \leq 1$, $f'(x) > 0$ and $f'(x) > 0$ for all $x \in \mathbb{R}$. This means that the function is strictly increasing. This means that if the function changes signs, it is impossible to change signs again. Therefore, there will be only one root.

1.3 Use the bisection code to solve for r accurate to 8 digits. Include the calling script, the resulting final approximation, and the total number of iterations used.

Using the values of x_1 and x_2 I guessed, the bisection code calculated the root as $r = 0.8878622154822129$ in 28 iterations.

```
f = lambda x: 2*x-np.sin(x) - 1
a = 0
b = np.pi
tol = 1e-8
```

2 The function $f(x) = (x-5)^9$, has a root with multiplicity 9 at $x = 5$, and is monotonically increasing (decreasing) for $x > 5$ ($x < 5$) and should be suitable for the function above. Use a=4.82, b=5.2, tol = 1e-4 and use bisection with:

2.1 $f(x) = (x-5)^9$

```
f = lambda x: (x-5)**9
a = 4.82
b = 5.2
tol = 1e-4
```

```
d =  5.01 , n =  0
d =  4.915 , n =  1
d =  4.9625 , n =  2
d =  4.98625 , n =  3
d =  4.998125 , n =  4
d =  5.0040625 , n =  5
d =  5.00109375 , n =  6
d =  4.999609375 , n =  7
d =  5.000351562500001 , n =  8
d =  4.9999804687500005 , n =  9
d =  5.000166015625 , n =  10
the approximate root is 5.000073242187501
f(astar) = 6.065292655789404e-38
```

2.2 The Expanded version

```
f = lambda x: (x**9 - 45*x**8 + 900*x**7 - 10500*x**6 +
        78750*x**5 - 393750*x**4 + 1312500*x**3 -
        2812500*x**2 + 3515625*x - 1953125)
```

```
    d =  5.01 , n =  0
d =  5.105 , n =  1
d =  5.1525 , n =  2
d =  5.12875 , n =  3
the approximate root is 5.12875
f(astar) = 0.0
```

2.3 Explain what is happening

The expanded form is more inaccurate because it has compounding errors, as found in HW1. In this case, the bisection method is failing because the code is registering that the error is less than the tolerance. Looking at the last iteration, the error from the actual root is 0.12875. This, when exponentiated, is very small, $9.7213e-9$. This means the code thinks it has reached the desired precision, and it stops calculating.That is why the expanded form is completely wrong and not within the desired margin.

3 Problem 3

3.1 Use a theorem from class (Theorem 2.1 from text) to find an upper bound on the number of iterations in the bisection needed to approximate the solution of $x^3 + x - 4 = 0$ in the interval $[1, 4]$ with accuracy of 10^{-3}.

$$|e_0| = |x_0 - r| < \frac{b-a}{2}$$
$$|e_1| = |x_1 - r| < \frac{b-a}{4}$$
$$\vdots$$
$$|e_n| = |x_n - r| = \frac{b-a}{2^n}$$

Bisection stops when $|e_n|$ <tol.

$$\frac{b-a}{2^{n+1}} < \text{tol}$$
$$2^{n+1} < \frac{b-a}{\text{tol}}$$
$$n > \log_2(\frac{b-a}{\text{tol}}) - 1$$
$$n > \log_2(\frac{4-1}{10^{-3}}) - 1$$
$$n > 9.96578428466$$
$$\boxed{n = 10}$$

This is the maximum iterations necessary to achieve the required precision. Any more iterations go past the required precision, and are unnecessary.

3.2 Find an approximation of the root using the bisection code from class to this degree of accuracy. How does the number of iterations compare with the upper bound you found in part (a)?

```
f = lambda x: x**3 + x - 4
a = 1
b = 4
tol = 1e-3
```

```
d =  2.5 , n =  0
d =  1.75 , n =  1
d =  1.375 , n =  2
d =  1.5625 , n =  3
d =  1.46875 , n =  4
d =  1.421875 , n =  5
d =  1.3984375 , n =  6
d =  1.38671875 , n =  7
d =  1.380859375 , n =  8
d =  1.3779296875 , n =  9
d =  1.37939453125 , n =  10
the approximate root is 1.378662109375
f(astar) = -0.0009021193400258198
```

The approximation was very accurate. The actual code required 10 iterations, compared to the predicted minimum of 10 iterations.

4 Problem 4

Definition 1: Suppose $\{p_n\}_{n=0}^{\infty}$ is a sequence that converges to p with $p_n \neq p$ for all n. If there exists constants λ, α such that

$$\lim_{n\to\infty} \frac{|p_{n+1}-p|}{|p_n-p|^{\alpha}} = \lambda$$

then $\{p_n\}_{n=0}^{\infty}$ converges to p with order α and asymptotic error constant λ. If $\alpha = 1$ and $\lambda < 1$, then the sequence converges linearly. If $\alpha = 2$, the sequence is quadratically convergent.

Which of the following iterations will converge to the indicated fixed point x_* (provided x_0 is sufficiently close to x_*)? If it does converge, give the order of convergence; for linear convergence, give the rate of linear convergence.

$$\lim_{n\to\infty} \frac{|p_{n+1}-p|}{|p_n-p|^{\alpha}} = \lambda$$

$$\log|p_{n+1}-p| - \alpha\log|p_n-p| = \log\lambda$$

$$\Rightarrow \alpha = \frac{\log\left(\frac{|p_{n+1}-p|}{|p_n-p|}\right)}{\log\left(\frac{|p_n-p|}{|p_{n-1}-p|}\right)}$$

$$\lambda \approx \frac{|p_{n+1}-p|}{|p_n-p|^{\alpha}} \text{ for large } n$$

4.1 $x_{n+1} = -16 + 6x_n + \frac{12}{x_n}, x_* = 2$

```
Alpha 1 =  1.0 , lambda 1 =  6.0
```

This sequence convergence with linear convergence since $\alpha = 1$. The rate of convergence is $\lambda = 6$.

4.2 $x_{n+1} = \frac{2}{3}x_n + \frac{1}{x_n^2}, x_* = 3^{1/3}$

```
Alpha 2 =  nan , lambda 2 =  nan
```

This sequence does not converge.

4.3 $x_{n+1} = \frac{12}{1+x_n}, x_* = 3$

```
Alpha 3 =  1.2892242269941028 , lambda 3 =  11328.94106922328
```

This sequence does converge with order $\alpha \approx 1.29$.

```
    import numpy as np
import matplotlib.pyplot as plt

# Sequences for parts a, b, c
def xn1(x):
    return -16 + 6*x + 12/x

def xn2(x):
    return 2/3*x + x**(-2)

```

```
def xn3(x):
    return 12/(1+x)

# Routine to calculate the sequence
def calcSequence(xn1,x0,N):
    p = []
    for x in range(N):
        p.append(x0)
        x = xn1(x0)
        x0 = x
    return(p)

# Initial guess and sequence length
N = 150
x01 = 2.01
x02 = 3**(1/3) + 0.01
x03 = 3 + 0.01

# Calculating sequence
p1 =  calcSequence(xn1,x01,N)
p2 =  calcSequence(xn2,x02,N)
p3 =  calcSequence(xn3,x03,N)

# Calculating alpha and lambda
p_actual1 = 2
p_actual2 = 3**(1/3)
p_actual3 = 3

n = 100
alpha1 = np.log(np.abs(p1[n+1]-p_actual1)/np.abs(p1[n]-p_actual1))/np.log(np.abs((p1[n]-p_actual1)/np.abs(p1[n-1]-p_actu
l1 = np.abs(p1[n+1]-p_actual1)/np.abs(p1[n]-p_actual1)**alpha1
print("Alpha 1 = ", alpha1,", lambda 1 = ", l1)

n = 10
alpha2 = np.log(np.abs(p2[n+1]-p_actual2)/np.abs(p2[n]-p_actual2))/np.log(np.abs((p2[n]-p_actual2)/np.abs(p2[n-1]-p_actu
l2 = np.abs(p2[n+1]-p_actual2)/np.abs(p2[n]-p_actual2)**alpha2
print("Alpha 2 = ", alpha2,", lambda 2 = ", l2)

n = 100
alpha3 = np.log(np.abs(p3[n+1]-p_actual3)/np.abs(p3[n]-p_actual3))/np.log(np.abs((p3[n]-p_actual3)/np.abs(p3[n-1]-p_actu
l3 = np.abs(p3[n+1]-p_actual3)/np.abs(p3[n]-p_actual3)**alpha3
print("Alpha 3 = ", alpha3,", lambda 3 = ", l3)

```

5 All roots of the scalar equation $x - 4\sin(2x) - 3 = 0$ are to be determined to 10 accurate digits.

5.1 Plot $f(x) = x - 4\sin(2x) - 3$. All the zeros should be in the plot. How many are there?

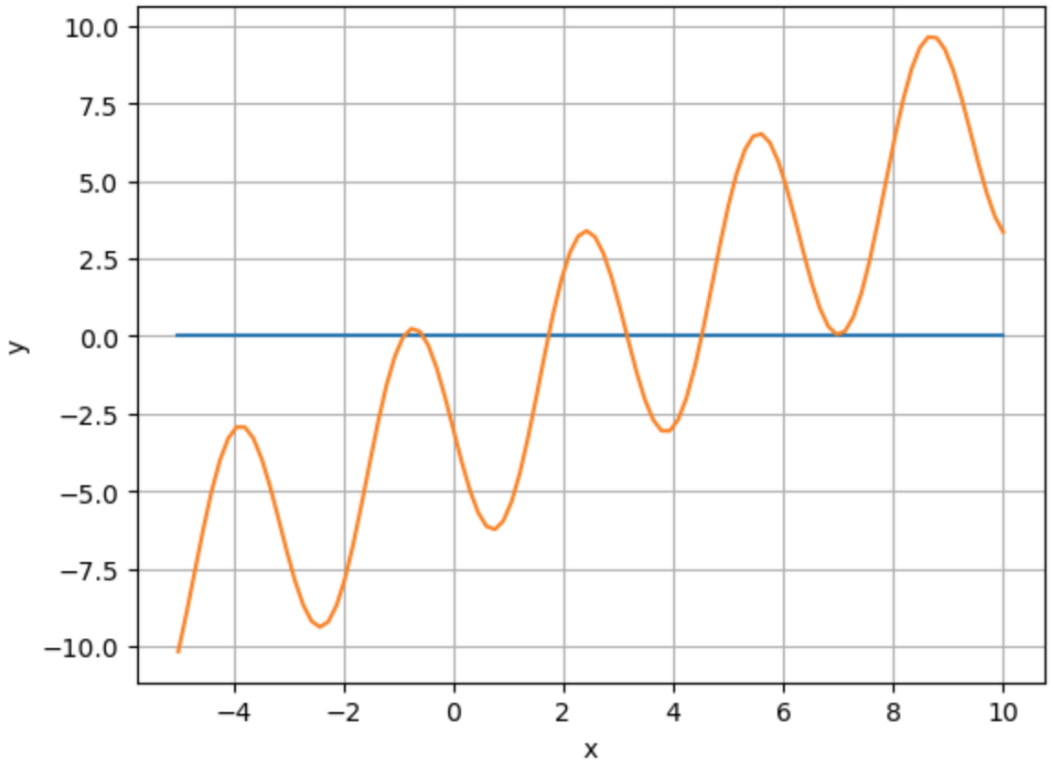

Figure 1: Plot of the function to determine approximate zeros and number of roots

This figure shows that there should be 6 roots. Outside of this domain, the value of x is far greater than the value of $-4\sin(2x)$, so the function follows linear behavior and no more roots will occur. Therefore, the 6 visible roots are all the roots of the function. It is also possible that there will be only 5 roots, since the rightmost potential root may or may not equal zero. Indeed, by zooming in on that point, it is possible to see that it is not a root. Thus, this function only has 5 roots.

5.2 Write a program to compute the roots using the fixed point method. Use a stopping criterium that gives an answer with 10 correct digits. You may have to change the error used in determining the stopping criterion. Give a theoretical explaination.

To perform Fixed Point Method to find the roots of this equation, you must rearrange the equation to be of the form $g(r) = r + f(r)$, where $f(r) = 0$. Rearranging the equation, you get

$$g(x) = 4\sin(2x) + 3$$

By using the fixed point method to find where this function intersects the function $y = x$. The intersections to this function will be the roots of the original equation.
In order to compute the roots, I used a brute force method, which is more computationally expensive, but it is more general. I tried writing some algorithms to determine the guesses x_0, however these required certain conditions for

the function that decrease the generality of this code. The way my code is written takes in initial bounds which theoretically cover all of the possible roots (found using the graph in figure 1), then it creates 1000 equally spaced points between these points. The value 1000 was chosen somewhat arbitrarily. If the function has very tightly packed roots, where the spacing between the roots is small, the value must be increased to ensure all points are captured. Then, this program iterates through all the points in the x guess vector, and returns any roots that are found.

```
    import numpy as np
import matplotlib.pyplot as plt


# BRUTE FORCE METHOD
tol = 1e-10
tries = 1000 # Number of possible points in [a,b] to try
Nmax = 100

# Bounds for guesses
a = -1
b = 8
pot_x = np.linspace(a,b,tries) # Potential values of x within a given range
solutions = []

g = lambda x: x-4*np.sin(2*x) - 3

# define routines
def bisection(f, a, b, tol):
    fa = f(a)
    fb = f(b);
    if (fa * fb > 0):
        ier = 1
        astar = a
        return [astar, ier]
    #   verify end points are not a root
    if (fa == 0):
        astar = a
        ier = 0
        return [astar, ier]
    if (fb == 0):
        astar = b
        ier = 0
        return [astar, ier]
    count = 0
    d = 0.5 * (a + b)
    while (abs(d - a) > tol):
        fd = f(d)
        #print("d = ",d,", n = ", count)
        if (fd == 0):
            astar = d
            ier = 0
            print(astar)
            return [astar, ier]
        if (fa * fd < 0):
            b = d
        else:
            a = d
            fa = fd
        d = 0.5 * (a + b)
        count = count + 1
```

```
    astar = d
    ier = 0
    return [astar, ier]


for i in range(tries):
    i = int(i)

    # Calculating the fixed point for all x in the potential x vector
    solution = bisection(g, pot_x[i-1],pot_x[i], tol)
    #print(pot_x[i-1]," < x0 < ",pot_x[i],", root = ",solution)
    if solution[1]==0 and not i == 0:
        solutions.append(solution[0])

print("Roots = ", solutions)


```

```
Roots =  [-0.8983565815262966, -0.5444424006718773, 1.732069502207073, 3.1618264865902095, 4.517789514148019
```